Bolt Beranek and Newman Inc.

bbn

AD A074728

12

Report No. 4181

LEVEL III

A069608

Research in Natural Language Understanding

Quarterly Progress Report No. 6, 1 December 1978 to 28 February 1979

D D C
RECEIVED
OCT 5 1979
E

DDC FILE COPY

Prepared for:
Defense Advanced Research Projects Agency

This document has been approved for public release and sale; its distribution is unlimited.

79 09 26 001

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER BBN Report No. 4181	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) RESEARCH IN NATURAL LANGUAGE UNDERSTANDING, ~~Quarterly Technical Progress Report No. 6~~ 1 December 1978 - 28 February 1979		5. TYPE OF REPORT & PERIOD COVERED Quarterly Progress Report
		6. PERFORMING ORG. REPORT NUMBER BBN Report No. 4181
7. AUTHOR(s) (10) William A./Woods		8. CONTRACT OR GRANT NUMBER(s) (15) N00014-77-C-0378, ARPA Order-3414
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, MA 02138 V 060100		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Department of the Navy Arlington, VA 22217		12. REPORT DATE (11) 28 Feb[redacted]79
		13. NUMBER OF PAGES 34
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(12) 41 (14) BBN-4181

18. SUPPLEMENTARY NOTES

(9) Quarterly technical progress rept. no. 6, 1 Dec 78-28 Feb 79,

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parallel machines, parallel algorithms, marker passing, situation recognition, situation-action rules, production rules, knowledge representation, structured inheritance networks.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discussed an abstract parallel machine organization and a class of "marker passing" algorithms for a family of important operations required for intelligent manipulation and use of knowledge. These operations have the characteristic that they involve a considerable amount of "non-deterministic" programming and/or search. The major such problem is that of "high level perception" or "situation recognition" - the problem of finding which of a large collection of situation-action rules are satisfied at any cont'd. → oua

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified. 060 100

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

79 09 26 001

Abstract (cont'd.)

given moment. The combinatoric costs of various algorithms for large collections of rules are discussed, and a proposed parallel architecture for performing such operations in real time is presented. An example marker passing algorithm for the problem of finding the most specific concepts that subsume a given input description is presented, and issues relative to the implementation of such algorithms on physical machines are discussed.

Accession For
NTIS GRA&I ☑
DDC TAB □
Unannounced □
Justification______
By______
Distribution/
Availability Codes
Dist | Avail and/or special
A

RESEARCH IN NATURAL LANGUAGE UNDERSTANDING

Quarterly Technical Progress Report No. 6

1 December 1978 - 28 February 1979

ARPA Order No. 3414	Contract No. N00014-77-C-0378
Program Code No. 8D30	Contract Expiration Date: 31 August 1979
Name of Contractor: Bolt Beranek and Newman Inc.	Short Title of Work: Natural Language Understanding
Effective Date of Contract: 1 September 1977	Principal Investigator: Dr. William A. Woods (617) 491-1850, x4361
Amount of Contract: $712,572	Scientific Officer: Gordon D. Goldstein

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 3414

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-77-C-0378.

TABLE OF CONTENTS

Parallel Algorithms for Real Time Knowledge Based Systems

W. A. Woods

1. Introduction

In various attempts to construct systems that automatically transform ordinary computer programs into programs that can capitalize on parallel execution, it is common to discover that the maximum effective parallelism that can be gained for a given program is quite limited (factors of 4 to 6 are not atypical). This is due to the fact that the potential parallelism that is exploited by such systems is essentially the parallel computation of subexpressions in an equation or the parallel evaluation of arguments to a function call. Such parallelism is limited to the number of subexpressions an equation has or the number of arguments in a function call, and these numbers are not large.

For computations that are typical of nondeterministic algorithms, however, the number of alternative computation paths that can be potentially followed in parallel can easily grow into the hundreds (or more). For many searches to find something satisfying a condition, one may find lists being scanned which are themselves hundreds long (even longer lists would be used if it were not for the computational cost). Often, the condition to be tested involves a pattern match which could itself have a large number of alternative choices (e.g., when the pattern contains one or more substring variables). Hence, in this domain, it is not unreasonable to expect factors of potential parallelism in the hundreds (or even thousands for very large data bases).

Most successful parallel processing architectures have resulted from designing an architecture to support some specific class of algorithm which is known to be important and/or expensive for some useful class of problems. Examples are Fast Fourier Transforms (for signal processing applications) and Key retrieval for associative memory processors. The former has turned out to be immensely important for a variety of signal processing applications, while the latter turns out to be extremely useful for storage management in virtual memory systems and presumably has many other useful applications as well.

However, contrary to what one might have expected, the intuitive attractiveness of an associative memory for various kinds of Artificial Intelligence applications and other sophisticated symbolic processing applications has not resulted in the widespread use of associative processors for these applications. The reason for this, I believe, is due to the fact that the kinds of retrievals required for most sophisticated applications are not merely direct retrieval of a known key, but rather the discovery of a stored item that matches a complex pattern. Some associative processors permit patterns that can be expressed as "don't care conditions" on certain bit positions in the key, but this is not nearly sufficient.

In this paper, I want to discuss an important class of operations for sophisticated symbolic computation that are of sufficient universality to make them worth considering for special architectural treatment. Appropriate algorithms and specialized hardware for these operations hold promise for capitalizing on potential parallelism to gain significant improvements in elapsed real time required to perform complex intelligent computations in knowledge based systems.

The class of problems that I want to consider are problems of what I have called "high level perception" or "situation recognition". The problem is to find all of the patterns in a data base of pattern schema that are satisfied by a given input, where the class of possible patterns is as general as possible. This basic recognition problem is a fundamental "inner loop" operation in almost all sophisticated "intelligent" symbolic processing applications, including language understanding, visual perception, medical diagnosis, mechanical inference, robot problem solving, automatic program synthesis, and the general class of knowledge based systems. The efficiency of this basic process is critical to any rule based system that is to operate with a large and sophisticated set of rules.

2. Situation Recognition Algorithms

In a system whose behavior is driven by a large set of rules, a major inner-loop operation is the determination of the set of rules that are applicable to a given situation. For sufficiently small sets of rules, this can be done by considering each rule in turn and matching its conditions against the current situation (e.g., matching a production rule against the contents of a specified set of registers). For larger sets of rules, this becomes impractical.

For a straightforward set of production rules, an efficient means of finding matching rules can be done by organizing a decision tree made up of the various tests that occur in the patterns of the production rules, and storing a list of the corresponding rules at each of the leaves of this tree. This significantly decreases the computation for determining the matching rules, but is based on an assumption that the pattern parts of the rules are essentially "flat" Boolean combinations of simple measurements on the current state. As the number and complexity of the rules increase, the decision tree method faces two alternative strategies. Either the decision tree can be interpreted as a deterministic automaton (with only a single path through the tree being followed), or it can be interpreted as a "nondeterministic" automaton (in which alternative paths through the tree can be explored simultaneously).

If the decision tree is to be a deterministic sequence of measurements leading to a unique list of matching rules, its depth will eventually begin to approximate the total number of measurements that are made in all the rules, although any given rule will actually make use of the results of only a small fraction of those measurements. Questions of optimality in the order of testing the conditions arise, but in general it will not be possible for a given rule to lie at the end of a path through the decision tree that tests only the conditions relevant to that rule. Instead, for most rules, there will be a large number of tests made in discovering its match that are done for the purpose of ruling out other possibilities and do not affect the applicability of the rule in question.

For each such "don't care" condition, there will be a bifurcation in the decision tree with each path leading to a possible match of the rule in question (but differing in what additional rules may be tested along the path). Thus, if there are n such don't care conditions in the tree before the last necessary condition is tested, there may be as many as 2**n different places in the decision tree where the rule in question needs to be stored (i.e., some pointer that accesses the action part of the rule needs to be stored).

The above argument is slightly simplified by the assumption that tests are done in essentially the same order on each of the paths through the decision tree, but the qualitative nature of the effect is still there even if the subsequent test and their order is made conditional on the results of previous tests. The upshot is that if there are n rules total, with say 5 elementary conditions to be tested per rule, and say 2n different elementary conditions total (i.e., on the average each rule shares approximately 3 of its conditions with other rules), there would be 2**2n possible combinations of conditions.

Since in general any given measurement will only make a difference for a few of the rules and will be a don't care condition for the others, most of the measurement decisions in the decision tree will not significantly reduce the number of measurements that will have to be made on the two alternative branches leading from that decision. It is probably generous then to estimate that the depth of the tree might be halved due to measurements that need not be made on some branches of the tree due to a previous measurement making them unnecessary. That would still leave 2**n combinations of conditions that would have to be explicitly represented in the decision tree. (Unfortunately, a miniscule difference between two subtrees is sufficient to keep them from being merged in a deterministic decision tree so there is little expected saving due to merging of common subtrees.)

Under these assumptions, a modest system of 1000 rules would require 2**1000 nodes in its decision tree (current address spaces are only 2**18 memory cells) and would require on the order of 1000 elementary conditions to be tested to determine the rule or rules applicable to a given situation (i.e., the inner loop decision of what rule to try next would require 1000 elementary measurements). This number of elementary measurements is not completely out of the question, although it is not attractive (especially since they have to be done sequentially) and would become even less so for 100,000 rules. What is critical, however, is the fact that the space to store this decision tree is too large (approximately 300 orders of magnitude larger than current address spaces).

So let us consider the alternative of a non-deterministic decision tree - that is, a tree each of whose branches corresponds to finding a particular result for a particular measurement, but which allows a number of different measurements to be made at a given point, and in particular allows alternative measurements that are not mutually exclusive (and will follow all consistent alternatives if several such measurements are satisfied). This can be thought of as an ATN or GTN grammar with no recursion and no use of registers (see Woods [1978c]). In such a tree structure, each rule can be made to live uniquely at the end of a chain of measurements corresponding to exactly the conditions of that rule (although it is possible for reasons of sharing common subtests that one may want to permit a few don't care measurements along such paths).

For such a structure, a system of n rules, with the characteristics discussed above, would have considerably fewer than 5n nodes and might require 2n or 3n elementary measurements to determine which rules were applicable (depending on how many of a rule's 5 measurements were satisfied on the average when the rule does not match completely). Storage is no longer a problem for this approach, and the number of elementary measurements required may either have gone up by as much as a factor of 3 or it could have gone down to the order of log(n) if very few of the potentially non-deterministic choice points actually involve following more than one case. In practice, the expected number of elementary measurements would probably be about the same as for the former case, although with considerable potential variance depending on the nature of the rules. The important consideration, however, is that in this formulation, no individual rule depends on a chain of measurements longer than about 5. Thus there is enormous potential for parallelism in pursuing the alternative rule matches through the decision tree.

Of the above three options (linear scan, deterministic decision trees, and nondeterministic decision trees), the non-deterministic decision tree seems clearly superior to the other two. It involves fewer elementary measurements than the straightforward testing of each rule in turn, and in some cases could reduce the number of evaluations from order n to order log(n). Moreover, the storage costs are no greater and could be less than that of the direct representation of each rule. Both the direct evaluation of each rule and the non-deterministic decision tree permit vast amounts of potential parallelism for systems with large numbers of rules.

The observation that the non-deterministic decision tree is essentially an ATN with no recursion and no registers raises the possibility that one might be able to do more complex rule patterns or have greater flexibility for compact expression of a total collection of rules by generalizing to unrestricted ATN's and making use of recursion and register setting. This is in fact the case. By the use of recursion and a device analogous to well-formed substring tables, it is possible to make constellations of tests into subroutines that need only be executed once even though they may be used on several different paths through the nondeterministic network. The use of registers will then permit a computation path to keep a record of the results of such subroutines. The use of registers can also permit the merger of two similar parts of the network into a single structure, with the differences between the two kept in the registers.

The above analogy to ATN grammars may seem strange to those accustomed to thinking of grammars only as devices for parsing strings of symbols. However, the ability to do arbitrary tests on an arc permit an ATN to be used essentially as a universal non-deterministic Turing machine. This ability has been formalized in Woods [1978c] into a generalization of ATN grammars called a "Generalized Transition Network" or GTN. A GTN permits the same kind of non-deterministic computation structure as an ATN, but is generalized to analyzing an arbitrary "perceptual domain". In a GTN, one characterizes the situations that can be induced from a set of rules, and then associates the right-hand sides of the rules with the situations in which they should be executed. Viewed in terms of GTN's, the problem of determining which rules to apply to a given situation is equivalent to parsing that situation into a characterization which then determines which rule actions are applicable.

Once this perspective is taken, it becomes clear that the use of a GTN to characterize situations also permits a more extensive classification of possible interactions among sets of simultaneously matching rules than merely a set of matching rules. For example, if one rule has a more specific pattern than that of another rule, then whenever the first rule matches, the second rule will also. If the rules are taken as merely a set of rules, then these two rules will compete with each other in the cases where the more specific rule matches. In the GTN formulation, it is possible to indicate that in certain situations one rule is to take precedence over another, while in others the effects of the two rules are to be combined, and in still others the rules should be considered as competing alternatives.

A previous report [Woods, 1978c] discusses motivations for introducing mechanisms of inheritance into the specification of a GTN for situation recognition applications. A structured inheritance network such as KLONE [Brachman, 1978] can be interpreted as specifying a GTN that provides such an inheritance mechanism and recognizes occurrences of the concepts described in the network. That is, a given concept with a particular set of roles with value restrictions and a particular set of structural conditions can be thought of as having an associated GTN (not necessarily explicitly constructed) whose states are characterized by subsets of the roles that have been filled and whose transitions correspond to filling an additional role. Such a network can be used as a "taxonomic lattice" [Woods, 1978b] to organize "advice" to be acted upon in different situations. One can imagine a process that simultaneously attempts to parse an input into all of the possible concepts that are known to the network. However, what one would actually like to find are the most specific concepts in the inheritance network that can parse the input (the more general concepts will then be inherited automatically). Thus an important algorithm for situation recognition is an algorithm that takes a given input situation and finds the most specific concepts in the system's taxonomy that are satisfied by the input situation.

3. Subsumption Algorithms

An inheritance network gains considerable conceptual efficiency from being able to store a learned rule or fact at a particular place in the taxonomy corresponding to the level of generality with which it holds. The development and maintenance of such a knowledge network requires an ability to introduce new levels of generality by operations of abstraction and specialization and an ability to move generic facts upward or downward in the taxonomy as learning proceeds. In the course of developing, maintaining, and using such a knowledge network, there are a class of operations that will be used extensively and which involve a large amount of computation. One of these is the operation just discussed, which finds the most specific concepts that are satisfied by a given input situation. The operations I am concerned with are all operations that perform the effect of a search of the concepts in the network to find concepts satisfying certain relationships to a given concept or description. They have the characteristic that one process spawns a number of relatively independent subcomputations, each of which may spawn a number of subcomputations, etc. I will refer to this class of operations as "bifurcation processes".

Bifurcation processes appear to be important for a wide variety of knowledge based systems. The bifurcation of independent subprocesses leads to large computational demands when executed on a serial computer, but has enormous potential parallelism. In this section, I will introduce a number of such operations that are of fundamental importance in a variety of symbolic computing activities and especially in applications of high level perception. In a later section, I will then discuss the potential for parallel implementation of these algorithms and their implications for special parallel architectures.

We have already discussed the operation of finding the most specific concepts in the network that subsume a given description.

Let us call this the MSS operation (for Most Specific Subsumer). Another related operation is an MGS routine (for Most General Specializer) which finds the most general concepts in the network that are subsumed by a given one. A somewhat more complex operation, but one that appears to be of central importance in high level perception is an operation I call MSMGU (for Most Specific Most General Unification). This operation implements a combination of a data base search and a unification operation.

The unification operation, which is the basic operation in Robinson's resolution method and a large class of mechanical inference systems, is essentially a *generalization* of a pattern match that effectively matches a pattern with another pattern and produces a pattern that will match whatever both of the original patterns would match (if such a "unified" pattern exists). This operation is used to match an inference rule with a schematic description of a state of a deduction to determine whether the inference rule is applicable. The operation of determining what rules to apply at a given point in a deduction consists of taking some set of rules from a data base and determining which of them can be unified with the current deduction state. Thus a basic operation is the discovery of those rules in the data base whose patterns are unifiable with a given description. The MSMGU operation is an operation to effectively search the data base for the most specific concepts that are unifiable with a given description, and to produce the most general unification of that description with each such concept.

The following is a list of important subsumption operations that we have identified so far, and there are undoubtedly many others.

- MSS (Most Specific Subsumer): Finds the most specific concepts in the network that subsume a given description.
- MGS (Most General Specializer): Finds the most general concepts in the network that are subsumed by a given description.
- INDS (Individuals): Finds the individual concepts in the network that satisfy a given description.
- MSU (Most Specific Unifiable): Finds the most specific concepts that are unifiable with a given description.
- MSMGU (Most Specific Most General Unifier): Finds or constructs the most general unifications of a given description with the most specific unifiable concepts that already exist in the network.

4. Marker Passing Algorithms

In order to capitalize on the potential parallelism for nondeterministic programming, it is necessary to be able to take a computation path and bifurcate it into multiple computation paths that share a considerable amount of context. This context-sharing causes a number of difficulties that stand in the way of realizing a computational advantage from the potential parallelism of such operations. If one makes separate copies of the shared context, then one has to pay the overhead of making those copies every time a computation path bifurcates (which may be very often). If on the other hand, one tries to share the context among genuinely parallel computations, then one has to provide for a multiple port memory sufficient to allow all of the potential processes to access the information. At the hardware level of most current memory architectures, this results in some combination of slowing down the access time of the memory by a factor equal to the number of simultaneously accessing processes (thus canceling most if not all of the advantage of the parallelism) and/or the duplication of the memory's address and access logic (which takes up a major portion, if not most, of the space on an LSI memory chip) for each of the separately accessing channels.

A possible direction to look for solutions to the above problem is to consider an architecture in which processor and memory functions are combined to some extent, thus distributing a significant amount of computation over the memory structure itself and localizing the interaction between process and memory. The computation model in this case is more like a network of communicating finite state machines than the conventional Von Neuman machine with a central processor and a separate memory.

Specifically, we will consider a hypothetical parallel machine consisting of a central controller and a collection of processing nodes, each of which contains a certain amount of permanent and scratch memory. Each node is connected to some number of neighbors by a communication channel called a link. A node can pass a message to another node along the link without contention with other communication links, subject only to the contention of several nodes simultaneously trying to send messages to a single destination. The central controller can broadcast messages to all nodes, and nodes can send messages to the controller through a contention network that resolves competition for the controller's attention according to priorities associated with the messages.

The abstract architecture that we envisage for this system is similar to that of [Fahlman, 1979], although we hypothesize a more powerful node than he does, as well as a greater ability for parallel, asynchronous activity without the detailed sequential supervision of the central controller. We will use the capabilities of the controller to regulate and modify the activities of the nodes and to make "policy" decisions on courses of action.

A node will be used to store a single fact, concept, rule pattern, procedure description (or part of one), etc., and it will be capable of the computation necessary to detect a pattern match of its pattern and participate in the pattern matching of other patterns of which its own pattern forms a subpart. The communication between nodes will take the form of messages of a particularly constrained format, somewhat comparable to that of a machine instruction for a computer with a two- or three-address instruction format. In place of addresses, this machine will use bit patterns called "marks", which will be considerably fewer in number than the number of nodes in the memory structure. Marks will be used in much the way that registers and temporary variables are used in conventional programming in that they can be reused for different purposes at different stages of a computation. However, instead of storing data or pointers to data in registers in a central processing unit, marks will be attached to the data in the memory nodes themselves.

The use of marks instead of registers has a number of advantages for realizing the potential parallelism of bifurcation processes:

(1) It reduces the communication bottleneck involved in a memory access to transfer a pointer from an arbitrary point in memory into a central register. An operation to move the "contents" of one mark to another consists of merely attaching an additional mark to those contents, without the necessity of getting a pointer to those contents into a central register and then storing the contents of that register in another place).

(2) It permits the simultaneous assignment of different values to the same mark and the parallel computation on the different hypotheses that correspond to those

different values. For example, if one wants to determine if any node with some condition A also has condition B, one can arrange all nodes with condition A to be marked with some mark m, and then simultaneously launch the computation for condition B for all nodes marked m.

(3) Finally, since the number of marks is smaller than the number of nodes, the messages being passed in such processes are smaller than the corresponding messages using pointers would be.

We will make the following assumptions: (1) The system will contain a finite number of marks (probably numbering a thousand or more) that can be associated with the nodes of the network in a number of different ways. (2) The system will contain a finite number of status codes (probably numbering a few dozen to a hundred) which can be used as "subscripts" on marks to indicate the way in which that mark is being used. (3) Each node will have an ability to store some number of messages, each consisting of a mark with a status code, or a pair of marks with status codes for each of the marks and also a status code for the pair as a whole.* (4) Such messages can also be passed along the links of the network. (5) A node can request a mark from a list of available ones.** (6) A node can send a message to the central controller. (7) The central controller can broadcast a message to all nodes of a given

* This may need to be generalized to include an ability to store triples of marks. Current experience, however, suggests it may be possible to get away with pairs.

** This may require freeing up and recycling an old mark.

type.*

Status codes associated as "subscripts" on a mark are used to indicate what purpose a mark is serving and to determine what effects that mark will have. There are a number of functions that can be served by marks with status codes. For example, if a mark is put on a node in status called OWN, then it can effectively serve as a temporary handle on that node. When so marked, the node could respond to broadcast requests for the owner of that mark. Coupled with the ability to respond to broadcast requests, this use of marks provides a way of implementing a selective addressing scheme. (This could obviate the need for a node to have an address decodable as a pointer in order to access it.)

Another use of marks is to construct temporary connections between different nodes in the network. For example, one can take the simultaneous presence of a given mark m on node A in status SOURCE and on node B in status DEST as a **virtual link** from A to B. If one wants to assign such a link specific properties, one can associate some node M with that link by assigning it the same mark m with status OWN and associate those properties with that node. Such virtual links make it possible to build up short-term memory

* Each type could have a separate broadcast channel, or each node could have an automatic detector that ignores broadcast messages whose type is different from that of the node. The system would have a small number of types (say from 1 to 6).

representations of potential network structure that may or may not be eventually incorporated into long-term memory. Such short-term "scratch" structures can be used in hypothesizing alternative interpretations of input sentences, planning future actions, making internal inference steps, etc. If nothing is done to make such virtual links permanent, they can be made to disappear when their marks become sufficiently old and are collected.

Pairs of marks can be associated with nodes to indicate some connection between the two marks. For example, a pair m1 s1 TRANS m2 s2 (referred to as a "trans" pair) can be used to indicate that if a node receives mark m1 in status s1 then it should store mark m2 in status s2 at that node. (Here the format for a pair of marks consists of the two marks followed by their status subscripts separated by the status code for the pair itself.)

A marker passing algorithm is expressed by determining a set of status codes to use for various purposes and a set of propagation rules for passing marks and pairs of marks through the network. An algorithm is then evoked by marking certain nodes in the network with statuses corresponding to its arguments and waiting for the consequences of those initial marks to propagate through the net. The algorithm "returns" its results by leaving certain nodes marked with a status indicating that they are the results. In some cases, the central controller may then access these results by broadcasting a request that they respond, but in many cases, the mere fact that they have been marked will constitute the input for another wave of marker propagation to compute some other inference.

Many applications of marker passing will be for problems of search through a network. In such cases, there may be priorities associated with alternative hypotheses, and for this reason one may want to associate priorities with messages being passed along the links in the network. For example, such priorities might be used to score the relative likelihoods of different possible semantic interpretations of a sentence. The priorities could then govern the access of the individual marker propagations to bottleneck resources such as requests from the central controller. The priorities could also affect the passage of marks through regions of the network that happen to get crowded by giving precedence to higher priority marks for service at a node. The use of priorities associated with a mark could thus affect the speed of propagation through the network and serve to automatically adjudicate any competition for machine resources among competing hypotheses in a search. As another example, higher priorities could be used on certain kinds of "cancel" marks in order to catch up with and terminate computations that have already been initiated but are subsequently discovered to be unnecessary due to the success of some other subcomputation.

5. The MSS Algorithm - An Example

To illustrate the use of marker passing algorithms to exploit the potential for parallelism in bifurcation processes, I will give here an example of a marker passing implementation of the MSS algorithm, used to find the most specific concepts in a KLONE network [Brachman, 1978] that subsume a given input description. This algorithm is one of the basic inferential operations on a KLONE network. It is used in assimilating new concepts into the network to determine what concepts the new concept should inherit information from. The most specific subsumers of a concept are the concepts to which it should be linked by superc links.

Assume that concepts can be described by list structured expressions characterized by the following simple BNF grammar:

```
<cspec> -> ( <cname> <rspec>* )
<rspec> -> ( <rname> <cspec>* )
```

where "cspec" stands for a concept specification and "rspec" stands for a role specification. In this notation, a concept is specified by a concept name (cname) designating a concept in the network under which the specified concept will fit, and a collection of role specifications that the specified concept must have. Similarly, a role specification will consist of the name of a parent role, plus concept specifications for value restrictions that the specified role must have. (This is a simplification of the range of information that is actually contained in a KLONE concept, but the simplification makes for a much more comprehensible example.) An example of a concept specification in this notation is:

```
(PERSON (HOBBY (GOLF)) (OCCUPATION (POLITICS)))
```

corresponding to the description "a person whose hobby is golf and whose occupation is politics".

A concept in a KLONE network is said to subsume such a description if anything satisfying the description would be an instance of that concept. For example the above description would be subsumed by the concept PERSON, and by concepts for people whose hobby is golf, people whose hobby is golf and whose occupation is politics, people whose hobbies include golf, people whose hobbies are outdoor sports, etc. The subsumption relationship between KLONE concepts and descriptions can be defined recursively by the conditions:

1) A concept subsumes a set of cspecs (i.e., subsumes the conjunction of those cspecs) if either (a) its name is the same as the cname of one of those cspecs, (b) it lies on a superconcept chain above a concept which satisfies condition a, or (c) all of its immediate superconcepts subsume the set of cspecs and each of its attached roles subsumes some rspec from some one of the cspecs.

2) A role subsumes an rspec if it has or inherits the name of the rspec and each of its value restrictions subsumes the set of cspecs of the rspec.

This recursive characterization of subsumption can be used as the basis of a strategy for assigning a given mark to all concepts in a network which subsume a given description. From there it is a simple additional step to propagate a "cancellation" mark up superc chains from each such concept, so that only the most specific subsumers have the subsumption mark and not the cancellation mark.

Recalling the basic structure of a marker passing algorithm, we need first to select a set of mark status codes to serve various purposes and then set up a set of propagation rules for passing marks through the network. For our example MSS algorithm, we will make use of the following status codes:

> **sub** -- a status used to mark a concept that subsumes a cspec or a role that subsumes an rspec. The MSS algorithm, when given a cspec and a mark m as arguments, will arrange to assign mark m with subscript sub to all concepts that subsume that cspec.
>
> **test** -- a status used to mark a concept which is a possible subsumer of a cspec depending on whether its other superconcepts and its roles satisfy the recursive conditions for subsumption.
>
> **found** -- a status that will be used to mark a role that has been determined to satisfy the recursive subsumption conditions.
>
> **trans** -- a status for a pair of marks that indicates that when the first mark of the pair is received, the second mark of the pair should be sent or stored.

The overall strategy of our MSS algorithm will be to start with a mark cm assigned to the particular cspec being processed (there may be any number of simultaneous MSS operations being carried out at any given time, and this mark will be used to keep the separate computations from mixing up their intermediate results). The mark cm will then be broadcast with subscript sub to the concept named by the cname of the cspec. This will cause the mark cm to propagate up superconcept chains from that node with subscript sub and to propogate down superconcept chains with subscript test. A mark rm will then be chosen (i.e., a new mark is requested from the central controller) for each of the rspecs of the cspec and a message rm found trans cm found will be broadcast to all roles in the network with the rname of that rspec, setting up a condition that will mark such a role as subsuming with respect to cm if all of its VR's send a message rm found (indicating that all the value restrictions of that role subsume cspecs in the rspec rm). Finally, for each of the cspecs of the rspec rm, MSS will be called recursively on that cspec using the mark rm, setting up conditions so that all concepts that subsume the cspecs of this rspec will eventually be marked with rm sub and will send a message rm found to their inverse VR's (i.e., the roles of which they are value restrictions).

An algorithm for injecting the necessary marks into the network can be written as follows:

```
 (DEFINE (MSS CM CSPEC)
     (BROADCAST TO CONCEPTS WHOSE NAME =
                (CNAME CSPEC) :  CM SUB)
     (FOR RSPEC IN (RSPECS CSPEC) DO
          RM <- (GETMARK)
          (BROADCAST TO ROLES WHOSE NAME =
                     (RNAME RSPEC) :
                     RM FOUND TRANS CM FOUND)
          (FOR CS IN (CSPECS RSPEC) DO
               (MSS RM CS) )))
```

where MSS is the function that initiates the MSS algorithm with mark CM for description CSPEC, BROADCAST is a function that broadcasts marks and mark pairs to roles and concepts (possibly restricted to have a specified name), and GETMARK is a function that requests a new mark from the controller.

In order for the signals injected by the above MSS function to have their effect, it is necessary that appropriate propagation rules be set up to pass marks along the links of the network, changing subscripts as appropriate, and interacting with other marks as indicated by trans pairs. A set of marker propagation rules for our example MSS algorithm is given below:

```
(WHEN A CONCEPT GETS M SUB
      AND DOESN'T HAVE M SUB :
      STORE M SUB,
      SEND TO SUPERC : M SUB,
      SEND TO SUBC : M TEST,
      SEND TO INVERSE VR : M FOUND)

(WHEN A CONCEPT GETS M FOUND,
      AND ALL OF ITS ROLE HAVE M SUB,
      AND ALL OF ITS SUPERC HAVE M SUB :
      SEND TO SELF : M SUB)
```

```
(WHEN A CONCEPT GETS M TEST, AND DOESN'T HAVE M SUB
      AND ALL OF ITS ROLE HAVE M SUB,
      AND ALL OF ITS SUPERC HAVE M SUB :
      SEND TO SELF : M SUB)

(WHEN A ROLE GETS M1 FOUND TRANS M2 FOUND,
      AND DOESN'T HAVE M1 FOUND TRANS M2 FOUND :
      STORE M1 FOUND TRANS M2 FOUND,
      SEND TO SUBROLE : M1 FOUND TRANS M2 FOUND)

(WHEN A ROLE GETS M1 FOUND,
      AND HAS M1 FOUND TRANS M2 FOUND,
      AND ALL ITS VR HAVE M1 SUB :
      STORE M2 SUB,
      SEND TO ITS CONCEPT : M2 FOUND)

(WHEN A ROLE GETS M1 FOUND TRANS M2 FOUND,
      AND ALL ITS VR HAVE M1 SUB :
      STORE M2 SUB,
      SEND TO ITS CONCEPT : M2 FOUND)
```

This set of propagation rules, together with the above MSS function for injecting marks into the network, will cause all those concepts in the network which subsume a given cspec to be marked with the mark assigned to that cspec in status sub. Notice that any number of simultaneous MSS computations can be going on in the network at any given moment, each with its own mark to keep it straight. In fact, the above MSS algorithm creates this kind of simultaneous activity in carrying out the MSS subsumption on the value restrictions of its roles. The elapsed real time required by this algorithm to compute a MSS is proportional to the longest chain of marker propagation necessary to complete the computation, which is in general much shorter than the number of machine instructions that would be required to compute the MSS operation on a sequential machine. This necessary elapsed time depends on the depth of the taxonomic lattice rather than on its breadth or on the total number of concepts. Moreover, the depth of the lattice will tend to be on the order of log(n) for a lattice of n concepts, so one would expect a real time requirement that grows on the order of the log of the number of concepts in the network. (The time required for MSS to inject the marks into the network is negligible since it is bounded by the number of elements in the input cspec, which is in general quite small.)

There are a number of problems that remain to be worked out in the specification and use of such algorithms. One of them is knowing how long to wait for the answer to develop. As a minimum criteria, one could wait until the network reaches a stable state, at which time all of the appropriate nodes will have been marked. However, in those cases where the results of this computation are to be passed on to other marker passing algorithms, it is not necessary to wait for the answer, but merely set up the propagation conditions so that as each node is marked as an answer, it automatically triggers the propagation of marks for the next computation to use that result. It is difficult to get an intuitive appreciation for the behavior of such a program, but the general characteristic would be that the overall answers to macroscopic questions would be found as soon as possible, with each subordinate computation being started as soon as its arguments were available. Final answers could then be reported to the central controller by a request that could be triggered by a node when it received a mark with a particular subscript.

One difficult problem in the use of such algorithms, which is a variation of the above "how long to wait" issue, has to do with the treatment of negation. For example, if we want to set up a condition so that all of the nodes that are marked m1 sub and do not have a subconcept that is also marked m1 sub (i.e. the most *specific subsumers) will be marked m1 mss*, we have to decide under what circumstances the negative condition will be assumed to be satisfied. There are basically two ways to deal with such conditions. One is to wait a specified amount of time for the contradicting mark to arrive to cancel the negative condition and if no such mark arrives in a reasonable amount of time, consider the negation satisfied. Another is to arrange a separate set of marker propagation conventions that will pass explicitly negative subscripts indicating that a condition is known not to be satisfiable and thus enabling the negative condition. There is probably need for both kinds of negation for various operations in such networks, so provision will need to be made for a node to start a countdown clock associated with a mark in order to capture the first kind of negation. (The second kind can already be done with the machinery introduced above by merely using a dual subscript not-sup which will be assigned to a concept that is proven not to subsume the cspec and using the presence of not-sup rather than the absence of sup as the condition in the rule.)

6. Possible Physical Realizations

The marker passing machines outlined above are essentially an abstract machine specification, amounting to a kind of cellular automata. There are a number of ways that one might go about creating a real physical machine to implement that abstraction. One way would be to implement a direct analog of the abstract machine with particular choices for parameters such as the number of bits in a mark, the number of nodes, the number of nodes adjacent to a given node, the amount of storage associated with a node, etc. However, it is difficult to say at this point what the appropriate choices for such parameters should be without some experience with actually writing programs and systems for such machines.

Another way to proceed is to construct a simulator or emulator for such a machine consisting of a collection of processors, each of which would manage some number of nodes of the abstract machine and would manage the message traffic between them. This approach has the flexibility necessary to vary the parameters of the abstract machine on a given physical machine, and also permits a tradeoff between the raw speed of the processors and the number of them required to achieve a given rate of throughput for a particular application. At the extreme end of this latter scale, is the simulation of the abstract machine on a single processor, which gives up all of the genuine parallelism advantages. However, even a single processor simulator can still gain some significant factoring advantages (see Woods [1978c]) for certain kinds of problems, and provides a way to begin exploring the problems of programming and debugging for such machines.

7. Conclusions

We have discussed several important operations required for intelligent manipulation and use of knowledge which have the characteristic that they involve a considerable amount of "nondeterministic" programming and/or search. That is, there is extensive use of operations of "hypothesize and test", "enumerate and filter", "scan for condition", or other kinds of search. The expense of these searches comes not so much from the fact that the lists being searched are long as from the fact that the tests to be applied to each choice are themselves composed of nested searches, which themselves may involve searches, etc. The result is a situation where the computational cost of the individual scans, each of which by itself would be reasonable, multiply together to become unmanageable when the patterns being searched for become complex and the data base of patterns being searched is of significant size.

A potential solution to the above combinatoric growth of computation cost is to capitalize on the fact that the different alternatives in a search are not logically dependent on each other and can therefore, in principle, be carried on independently. This solution involves trading the elapsed time required for searching the space of alternative possibilities for a combination of elapsed time and simultaneous use of multiple processors. We have been exploring a class of marker passing algorithms which hold promise for realizing this potential parallelism as an actual improvement in elapsed time.

In this paper, we have discussed the advantages of such parallel implementation of nondeterministic algorithms, we have outlined the basic architecture of an abstract marker passing machine for exploiting such parallelism, and we have given an example of a marker passing algorithm for such a machine and discussed the implementation of physically parallel hardware to carry out such algorithms.

8. References

Brachman, R.J. 1978. On the Epistemological Status of Semantic Networks. Report No. 3807, Bolt Beranek and Newman Inc., Cambridge, MA. April.

Fahlman, S.E. 1979. A **System for Representing and Using Real-World Knowledge**. Cambridge, MA: MIT Press.

Woods, W.A. 1978a. Marker Passing Algorithms. In Research in Natural Language Understanding, Quarterly Progress Report No. 2, BBN Report No. 3797, February.

Woods, W.A. 1978b. Taxonomic Lattice Structures for Situation Recognition. In TINLAP-2, Conference on Theoretical Issues in Natural Language Processing-2, University of Illinois at Urbana-Champaign, July 25-27.

Woods, W.A. 1978c. Generalizations of ATN Grammars. In Research in Natural Language Understanding, Quarterly Progress Report No. 4, BBN Report No. 3963, August.

Official Distribution List
Contract N00014-77-C-0378

	Copies
Defense Documentation Center Cameron Station Alexandria, VA 22314	12
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2
Office of Naval Research Code 200 Arlington, VA 22217	1
Office of Naval Research Code 455 Arlington, VA 22217	1
Office of Naval Research Code 458 Arlington, VA 22217	1
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1
Office of Naval Research New York Area Office 715 Broadway - 5th Floor New York, NY 10003	1

Naval Research Laboratory 6
Technical Information Division
Code 2627
Washington, D.C. 20380

Naval Ocean Systems Center 1
Advanced Software Technology Division
Code 5200
San Diego, CA 92152

Dr. A.L. Slafkosky 1
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D.C. 20380

Mr. E.H. Gleissner 1
Naval Ship Research & Development Center.
Computation & Mathematics Dept.
Bethesda, MD 20084

Capt. Grace M. Hopper 1
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D.C. 20350

Mr. Kin B. Thompson 1
NAVDAC 33
Washington Navy Yard
Washington, D.C. 20374

Advanced Research Projects Agency 1
Information Processing Techniques
1400 Wilson Boulevard
Arlington, VA 22209

Capt. Richard L. Martin, USN 1
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501

Director, National Security Agency 1
Attn: R53, Mr. Glick
Fort G. G. Meade, MD 20755